A Framework and Methods for Dynamic Scheduling of a Directed Acyclic Graph on Multi-core

Uma B[1], C R Venugopal[2]
[1] Malnad College of engineering, Hassan, India
Email: umaboregowda@gmail.com
[2] Sri Jayachamarajendra College of engineering, Mysore, India
Email: venu713@gmail.com

***Abstract*—The data flow model is gaining popularity as a programming paradigm for multi-core processors. Efficient scheduling of an application modeled by Directed Acyclic Graph (DAG) is a key issue when performance is very important. DAG represents computational solutions, in which the nodes represent tasks to be executed and edges represent precedence constraints among the tasks. The task scheduling problem in general is a NP-complete problem[2]. Several static scheduling heuristics have been proposed. But the major problem in static list scheduling is the inherent difficulty in exact estimation of task cost and edge cost in a DAG and also its inability to consider and manage with runtime behavior of tasks. This underlines the need for dynamic scheduling of a DAG. This paper presents how in general, dynamic scheduling of a DAG can be done. Also proposes 4 simple methods to perform dynamic scheduling of a DAG. These methods have been simulated and experimented using a representative set of DAG structured computations from both synthetic and real problems. The proposed dynamic scheduler performance is found to be in comparable with that of static scheduling methods. The performance comparison of the proposed dynamic scheduling methods is also carried out.**

***Keywords*: DAG, static scheduling, dynamic scheduling**

I. Introduction

The recent advancements in VLSI technology has led to increase in number of processing units on a single die i.e. multi-core. The trend to increase clock speed has hit a wall due to increased heat dissipation, increased power requirements and increased leakage current. The existing system software is inadequate for this new architecture and the existing code base will be incapable of delivering expected increased performance on this new architecture. They may not even be capable to harness the full potential of multi-core processor. Hence the need of new system software to exploit the complete power of multi-core.

A computational solution can be represented as a DAG, in which the nodes represent code segments to be executed. An edge from node 'u' to node 'v' indicates that output of code segment at node 'u' is an input to code segment at node 'v'. The weight of a node represents the execution time(estimated) of the corresponding code segment. The weight of edge (u,v) indicates the volume of data to be communicated from node 'u' to node 'v'. The objective of scheduling DAG structured computation is to minimize the execution time by allocating tasks to cores, while preserving precedence constraints[1].Scheduling methods can be broadly classified into static and dynamic methods. Several static task scheduling methods are found in literature[1,3,4]. Few drawbacks of static scheduling are a) need to correctly capture the program behavior as a DAG which is very difficult b) to know the exact weight of every node and edge c) size of the DAG depends on size of problem being solved, which grows enormously as problem size is increased d) can be applied to a limited class of problems. Dynamic scheduling overcomes these drawbacks, but has the little overhead of scheduling costs during runtime.

Although the benefits of dynamic data-driven scheduling were discovered in early 80's, recently they are being rediscovered as tools for exploiting the performance of multi-core processors. Our contributions in this paper include a)implemented a simulating framework to experiment dynamic scheduling methods b)implemented 4 methods of dynamic DAG scheduling c) compared the performance of proposed dynamic scheduling methods with static scheduling methods d) compared the performance of the proposed dynamic scheduling methods and analyzed their performance e) generated DAGs of well known linear methods like GE, LU, Laplace & FFT to be used as input to the proposed scheduler. The rest of the paper is organized as follows: In section 2 we provide the background of task scheduling. Section 3 gives a summary of the work done so far in scheduling. The proposed framework for dynamic DAG scheduling is given section 4. The kind of inputs tested is indicated in section 5. The performance of the schedulers is discussed in section 6. Section 7 concludes the paper and gives future direction of work.

II. Background

In our case, the input to the scheduler is a DAG. A task can begin its execution only after all its predecessors have completed execution. Static scheduling methods can be further grouped into list scheduling methods, clustering algorithms, guided random search methods and task duplication based algorithms. List scheduling methods basically involve 3 steps – a) prioritizing tasks b) assigning tasks to cores based on some criterion c) ordering the tasks assigned to a core. Unlike static scheduling methods, dynamic scheduling does not require accurate information on DAG, rather mapping of tasks to cores takes place on-the-fly. Also it can tolerate error in weight estimation

DOI: 01.IJIT.02.01.528

and more fault tolerant.

III. Related work

The scheduling problem has been extensively studied. A lot of static scheduling methods have been proposed. List scheduling methods are the generally preferred static scheduling methods for its low complexity and reasonably good quality schedules. HLFET[5](Highest Level First with Estimated Time) is a simple and low complexity method, which uses static level to prioritize tasks and uses earliest start time to assign task to cores. ETF[5] (Earliest Time First) selects the best pair of task-core, based on earliest execution start time, at each scheduling step. This method generates better schedules, but its time complexity is more. CPOP[1](Critical Path on Processor) underlines the importance of tasks lying on critical path, schedules them onto a separate processor, so that all critical tasks are executed at the earliest. The MCP(Modified Critical Path) algorithm[13] assigns the priority of a task node according to its As Late As Possible (ALAP) value. The ALAP of a task node is a measure of how far the node's start time can be delayed without increasing the schedule length. Qin Zheng [14] have studied to improve the performance of static algorithms when tasks execution times in a DAG are uncertain. The initial schedule, based on the estimated task execution times, is revised or modified (adapted) during runtime as necessary when tasks overrun or under run. The objective is to minimize the response time and the number of adaptations. Most list scheduling methods are non-preemptive. Fast Preemptive scheduling[15] method is more realistic and have lot of potentials. It has lot of flexibility in scheduling as low priority task can be executing until a high priority task becomes ready. This leads to better utilization of resources. Several other methods are proposed based on clustering, look-ahead mechanism and back filling approaches to schedule an application.

Dongarra et. al. [6] have presented a dynamic task scheduling approach to execute dense linear algebra algorithms on multi-core systems, also have provided analytical analysis to show why the tiled algorithms are scalable. The coarse-grain data flow model is the main principle behind emerging multi-core programming environments such as Cilk/Cilk++ [9], Intel R Threading Building Blocks (TBB) [9] and Tasking in OpenMP 3.0 [10]. All these frameworks use a very small set of extensions to common programming languages and involve a relatively simple compilation stage but potentially much more complex runtime system. Cilk[7] developed at MIT uses work-stealing policy to schedule the tasks, but the programmer has to explicitly identify parallel tasks using keywords. Laura De Giusti et al.[8] have developed a model for automatic mapping of concurrent tasks to processors applying graph analysis for the relation among tasks, in which processing and communicating times are incorporated. Viktor K. Prasanna et. al. [11] have implemented a lightweight scheduling method for DAG structured computations on multi-core processors. The scheduling activities are distributed across the cores and the schedulers collaborate with each other to balance the workload. We have made an attempt to adopt the static list scheduling methods to dynamically schedule a DAG.

IV. Basic framework of Dynamic Scheduler for a DAG

This section provides the overall view of the proposed 2-level dynamic scheduler and its basic working. Also, the different scheduling policies tried at both global and local level are discussed.

The input to our scheduler is a DAG. Scheduling happens at 2 levels – global and local level. The complete information about all tasks in the DAG is stored in the global scheduler. The job of this scheduler is to assign tasks to different cores and to update the status of related tasks, with an aim to reduce the overall time to complete the execution of DAG. Ready tasks are the ones, whose predecessors in DAG have completed execution and hence they are ready for execution. A local scheduler running on each core, maintains the set of ready tasks assigned to it by the global scheduler. It picks the task from the list in some order, so as to maximize scheduling objective and executes it. The completion of a task is informed to the global scheduler. Different Scheduling policies are tried at both global and local level.

Global scheduling policies :

Few policies proposed to assign the ready tasks to cores are discussed below.

1. The idea is that if all cores are busy without idling due to non-availability of data or tasks, then the best performance in terms of reduced makespan and high speedup can be expected. One simple way to do this, is balancing the load across all cores. To achieve this, an estimation of the load on all cores is maintained and the next ready task is assigned to the least loaded core at that point of time. This is a simple method and yet gives reasonably good schedules.
2. The drawback of the above load balancing policy is that the load may be balanced across the cores, but still the cores may be idling, because of waiting for data from other tasks executed on different cores. Depending on 1. the interconnection between the cores, the time required to transfer data between any two cores varies. This was not taken into consideration in the previous policy. Hence to reduce this idle time due to the non-availability of data, the ready task is assigned to a core, on which it will have the earliest start time. Thus the policy is to find the best pairing of task and core, among all ready tasks, so as to reduce the overall completion time. This method is better than the previous one, but its time complexity is more. One drawback is, it does the best mapping with the information on hand at that time, but does not look beyond i.e. off springs of the ready task, to take much better decision.

Local scheduling policies :

Simple methods used to pick one of the assigned tasks, at each core are given below.

1. One method is to execute the tasks in the order they are assigned to core. Though it is simple, it works out well, mostly

DOI: 01.IJIT.02.01.528

in cases where the number of waiting tasks at each core is less in number. This will be the case, when the degree of parallelism of DAG is less or the number of cores used is more in comparison to the degree of parallelism of DAG. Simplicity pays off in such cases.

2. Static level of a node indicates how far it is from the exit node. It is generally better to first schedule the nodes farthest away from the exit node. Hence the strategy is to pick the task with highest static level from the local list. The static levels of all nodes are computed and stored as a part of DAG, which is the input to dynamic scheduler.

3. If a node with more children are scheduled early, there are good chances that those children nodes may become ready nodes. Thus with these more number of ready nodes , there are better chances of mapping the right task to the right core, which in turn will reduce the makespan. This method works well, but not for all cases.

The different combinations of the above mentioned local and global policies are tried. Also, the dynamic scheduling is compared with the quality of schedule generated by static scheduler. To experiment, 2 static schedulers HLFET and ETF are implemented. In HLFET, the nodes are prioritized with static level. The task with the next highest priority is assigned to core on which it will have earliest start time. The complexity of this is $O(pv^2)$. In ETF, the nodes are prioritized using static level as in HLFET, but selects the best node-core pair, which gives the earliest start time. This generates better schedules but with time complexity of $O(pv^3)$.

V. Experimental Setup

This section discusses the performance metrics used and the different kind of DAG inputs used to test the proposed schedulers.

A. Comparison metrics:

The comparison of the algorithms are based on the following metrics.

Makespan: is the overall completion time of the DAG. Also known as schedule length.

Scheduling Length Ratio(SLR): The main performance measure is makespan. But since large set of DAGs with varying properties are used, it is better to normalize the obtained schedule length to the lower bound on schedule length. The lower bound is the critical path length. SLR is defined as the ratio of obtained schedule length to the critical path length.

Speedup: is the ratio of sequential time of executing DAG to the obtained schedule length. The sequential time of DAG is obtained by adding the cost of all nodes.

Efficiency: It is the ratio of obtained speedup to the number of cores used.

B. Test suite

The input DAG to the scheduler are chosen from

1. random DAGs from Standard Task Graph Set [12], from where many researchers take DAGs for comparing the performance of schedulers. The size of chosen DAGs are 50,100, 300 and 500 nodes.

2. DAGs of linear algebra methods including Gauss-Elimination(GE), LU decomposition (LU), Laplace and Fast Fourier Transform(FFT). A software was written to generate the DAGs of GE, LU, Laplace and FFT, given the size of matrix.

VI. Experiment Results and Analysis

This section presents the performance comparison of the proposed dynamic schedulers and static schedulers. 2 static schedulers namely HLFET and ETF are implemented. A total of 5 Dynamic scheduler with different combinations of stated global and local scheduling policies is experimented. For convenience purpose, dynamic scheduling algorithms are given below names.

load-fifo - load balancing at global scheduler and fifo order at local scheduler

load-sl - load balancing at global scheduler and using static level at local scheduler

est-fifo - best pair of task-core based on earliest start time at global scheduler and fifo order at local schdeuler

est-sl - best pair of task-core based on earliest start time at global scheduler and using static level at local scheduler

load-maxdeg - load balancing at global scheduler and choosing task with maximum out degree

The performance of 2 static schedulers and proposed 4 dynamic schedulers is compared, using random DAG from STG website[12]. Schedulers were run for different size of DAG, and for each size, several pattern of DAGs were tried. The average SLR of the makespan obtained for each size was recorded. The plot of SLR vs DAG size is given in Fig.1. There is a considerable variation in SLR obtained for different size of DAGs, because the schedule generated depends on the kind of DAGs. As the DAGs are random, there is variation in the schedules obtained. But the time taken by both static and dynamic schedulers are very close to each other. Since DAGs are random, the time required also varies, but in most cases, performance of dynamic scheduler is comparable to that of static ones. Thus dynamic schedulers can be used in place of static schedulers, with the added advantage that it can better tolerate inaccuracies in estimation of node cost and edge cost.

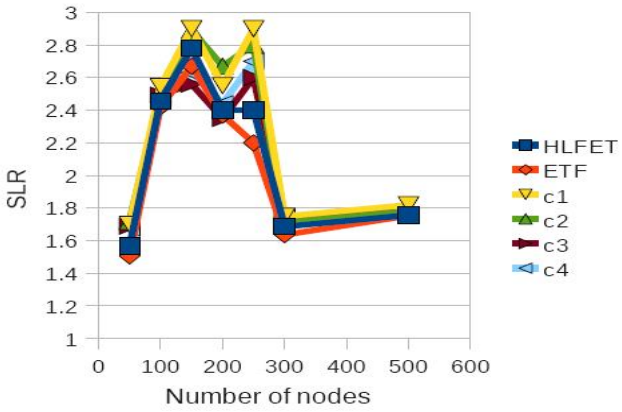

Fig 1. SLR for Random DAGs

DOI: 01.IJIT.02.01.528

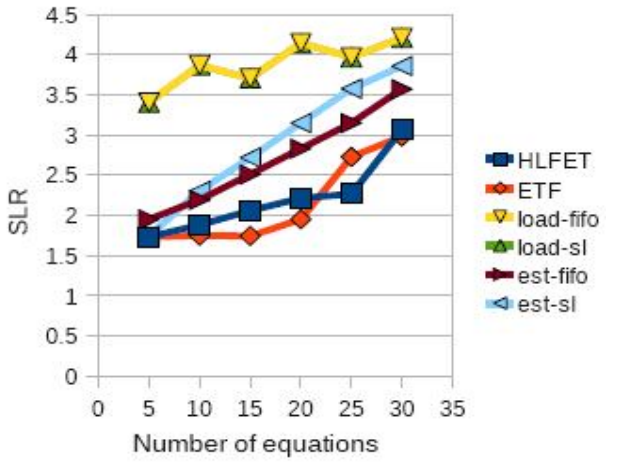

Fig 2. SLR for GE DAG

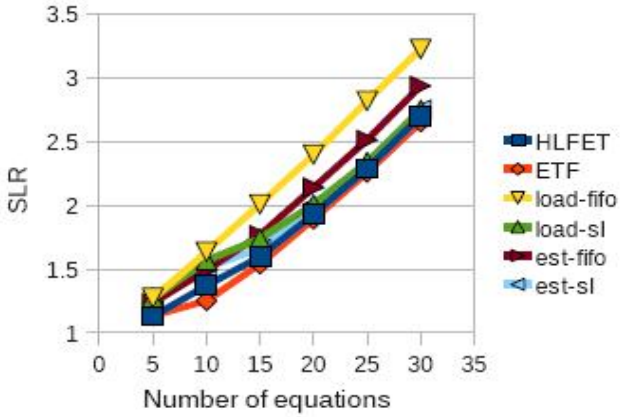

Fig 3. SLR for LU DAG

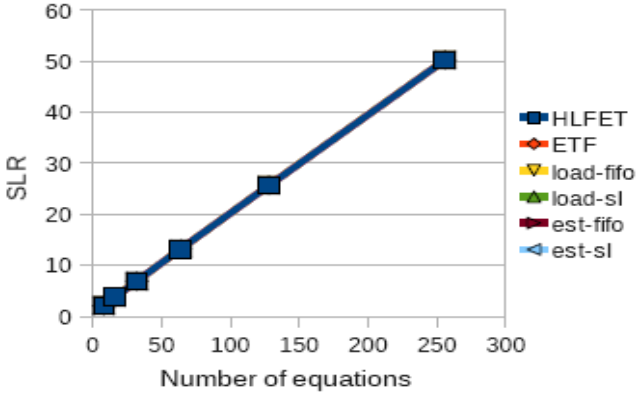

Fig 4. SLR for FFT DAG

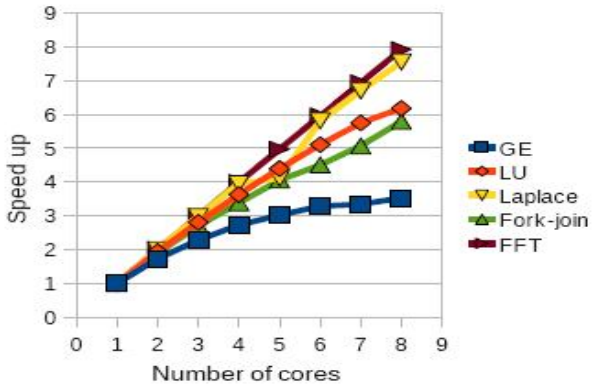

Fig 5. Speedup vs # of cores

The proposed dynamic schedulers were also found to perform well for widely used linear algebra algorithms like Gauss elimination(GE), LU-decomposition method, Laplace and FFT. A program was written to generate DAG of the above mentioned linear algebra methods, for any given input size. Thus the schedulers were tested for varied size of each linear algebra method. Fig 2. gives the performance both static and dynamic scheduler for GE method. The SLR for different number of equations is plotted. The simple load-fifo performance is slightly inferior to others. Because just load balancing is not sufficient in scheduling the tasks. The est-fifo scheduler outperforms other dynamic schedulers. This is because of good mapping decision at global level, but at the cost of added complexity.

Fig 3. shows the performance of schedulers for LU method. It is again evident that the proposed dynamic schedulers are comparable to that of static schedulers. The simple load-fifo scheduler is marginally inferior to others. For FFT and Laplace, all schedulers performance is almost the same. This is shown in Fig 4. for FFT. This is because that DAGs are regular and symmetric, it does not matter much which tasks is scheduled on which core. The computation and communication costs are almost same, hence the selection of task on a core does not affect the makespan considerably. So the performance is almost the same for all schedulers.

The scalability of the proposed schedulers is also measured. To show this, speedup is plotted for varying number of cores in Fig 5. Except for GE DAGs, for all other DAGs, schedulers show linear speed up. This is because only GE DAGs have high CCR, hence they can not well utilize the increased number of cores. With these experiments, it is quite clear that the proposed dynamic schedulers performance is comparable to that of static schedulers. This is proved for both synthetic and real application DAGs.

VII. Conclusion

In this paper, a framework to implement dynamic scheduler is proposed to schedule a given DAG. Three dynamic scheduling methods are proposed. The working of dynamic scheduler is simulated and its performance is found to be in comparable to that of well known static scheduler HLFET and EFT. The scheduling takes place at global and local level. Different strategies are tried at both levels. The performance of the proposed dynamic schedulers are compared. The proposed dynamic scheduler performance is found to be in comparable with static scheduler. The proposed dynamic est-sl scheduler marginally outperforms other dynamic schedulers.

Future work

We now have simulated dynamic scheduler with dummy tasks. With this promising results obtained, we shall extend dynamic scheduler to be actually implemented on real machine and to use real tasks instead of dummy tasks. Another direction of work is to explore other strategies for scheduling at both local and global level.

DOI: 01.IJIT.02.01. 528

REFERENCES

[1] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys, 31(4):406–471, 1999

[2] Cassavant. T. and J.A. Kuhl, 1988. Taxonomy of scheduling in general purpose distributed memory systems. IEEE Trans. Software Engg., 14: 141-154.

[3] E. Ilavarasan and P. Thambidurai , 2007.Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments , Journal of Computer Sciences 3 (2): 94-103, 2007

[4] T. Hagras, J. Janeèek . Static vs. Dynamic List-Scheduling Performance Comparison , Acta Polytechnica Vol. 43 No. 6/2003

[5] Kwok Y., Ahmed I. Benchmarking the Task Graph Scheduling Algorithms. Proc. IPPS/SPDP, 1998.

[6] F. Song, A. YarKhan, and J. J. Dongarra. Dynamic task scheduling for linear algebra algorithms on distributed- memory multi-core systems. Technical Report UT-CS-09-638, Computer Sciencee Department, University of Tennessee, 2009.

[7] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. "Cilk: An efficient multi threaded runtime system. In Principles and Practice of Parallel Programming", Proceedings of the fifth ACM SIG- PLAN symposium on Principles and Practice of Parallel Programming, PPOPP'95, pages 207– 216, Santa Barbara, CA, July 19-21 1995. ACM. DOI: 10.1145/ 209936.209958.

[8] Laura De Giusti et. al., " A Model for the Automatic Mapping of Tasks to Processors in Heterogeneous Multi-cluster Architectures" , JCS&T Vol. 7 No. 1 April 2007

[9] Intel Threading Building Blocks. http://www. threadingbuildingblocks.org/

[10] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.0, 2008. http://www.openmp.org/p-documents/spec30.pdf.

[11] Y. Xia and V. K. Prasanna, "Collaborative scheduling of dag structured computations on multi-core processors," in Conference Computing Frontiers, 2010, pp. 63–72.

[12] http://www.kasahara.elec.waseda.ac.jp/schedule/

[13] M. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," IEEE Trans. Parallel and Distributed Systems, vol. 1, pp. 330–343, July 1990.

[14] Qin Zheng, "Dynamic adaptation of DAGs with uncertain execution times in heterogeneous computing systems", IPDPS Workshops 2010 .

[15] Maruf Ahmed, Sharif M. H. Chowdhury, Masud Hasan, "Fast Preemptive Task Scheduling Algorithm for Homogeneous and Heterogeneous Distributed Memory Systems". SNPD 2008: 720-725

[16] H. Orsila, T. Kangas, T. D. Hminen, "Hybrid Algorithm for Mapping Static Task Graphs on Multiprocessor SoCs", International Symposium on System-on-Chip (SoC 2005), pp. 146-150, 2005.

DOI: 01.IJIT.02.01. 528